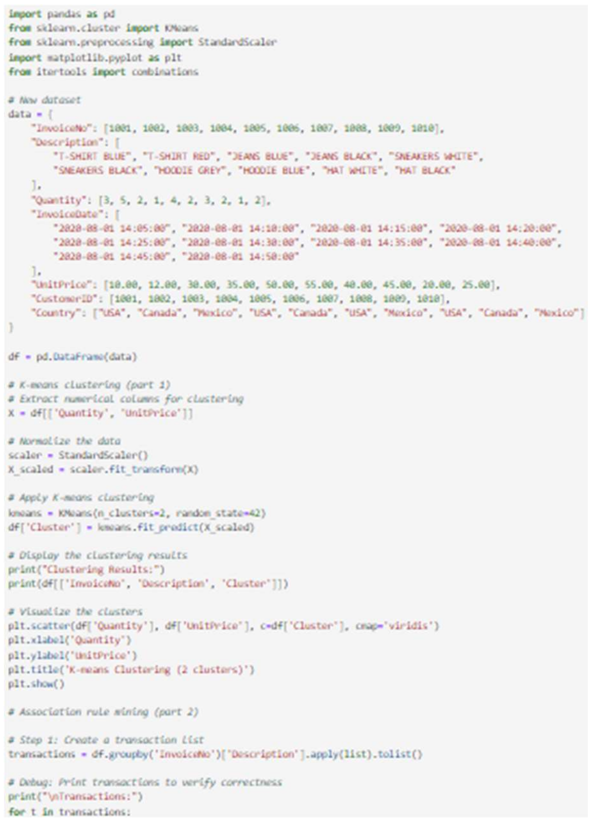
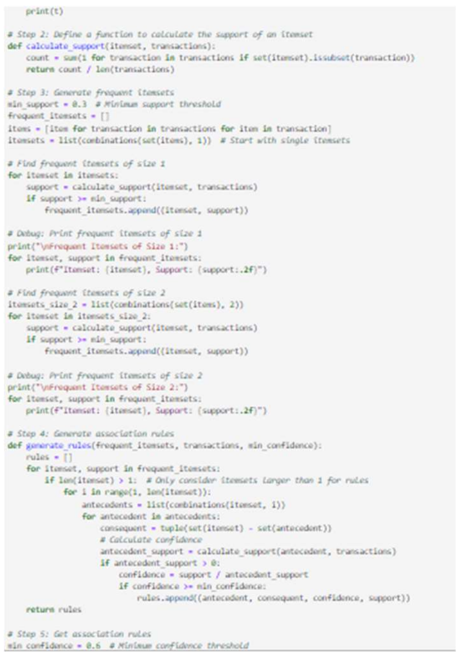
HOME ASSIGNMENT:

- **InvoiceNo**: A unique identifier for each transaction.
- **Description**: The product name or description.
- **Quantity**: The number of items purchased in a particular transaction.
- **InvoiceDate**: The date and time when the transaction occurred.
- **UnitPrice**: The price of a single unit of the item.
- **CustomerID**: A unique identifier for the customer who made the purchase.
- **Country**: The country where the customer is located.

  - **CODE:**



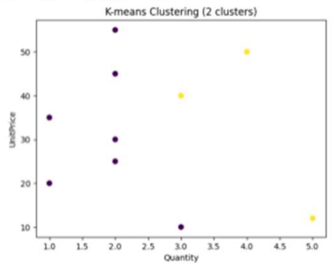
```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from itertools import combinations

# New dataset
data = {
    "InvoiceNo": [1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010],
    "Description": [
        "T-SHIRT BLUE", "T-SHIRT RED", "JEANS BLUE", "JEANS BLACK", "SNEAKERS WHITE",
        "SNEAKERS BLACK", "HOODIE GREY", "HOODIE BLUE", "HAT WHITE", "HAT BLACK"
    ],
    "Quantity": [3, 5, 2, 1, 4, 2, 3, 2, 1, 2],
    "InvoiceDate": [
        "2020-08-01 14:05:00", "2020-08-01 14:10:00", "2020-08-01 14:15:00", "2020-08-01 14:20:00",
        "2020-08-01 14:25:00", "2020-08-01 14:30:00", "2020-08-01 14:35:00", "2020-08-01 14:40:00",
        "2020-08-01 14:45:00", "2020-08-01 14:50:00"
    ],
    "UnitPrice": [10.00, 12.00, 30.00, 35.00, 50.00, 55.00, 40.00, 45.00, 20.00, 25.00],
    "CustomerID": [1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010],
    "Country": ["USA", "Canada", "Mexico", "USA", "Canada", "USA", "Mexico", "USA", "Canada", "Mexico"]
}

df = pd.DataFrame(data)

# K-means clustering (part 1)
# Extract numerical columns for clustering
X = df[['Quantity', 'UnitPrice']]

# Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply K-means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Display the clustering results
print("Clustering Results:")
print(df[['InvoiceNo', 'Description', 'Cluster']])

# Visualize the clusters
plt.scatter(df['Quantity'], df['UnitPrice'], c=df['Cluster'], cmap='viridis')
plt.xlabel('Quantity')
plt.ylabel('UnitPrice')
plt.title('K-means Clustering (2 clusters)')
plt.show()

# Association rule mining (part 2)

# Step 1: Create a transaction list
transactions = df.groupby('InvoiceNo')['Description'].apply(list).tolist()

# Debug: Print transactions to verify correctness
print("\nTransactions:")
for t in transactions:
```



```
    print(t)

# Step 2: Define a function to calculate the support of an itemset
def calculate_support(itemset, transactions):
    count = sum(1 for transaction in transactions if set(itemset).issubset(transaction))
    return count / len(transactions)

# Step 3: Generate frequent itemsets
min_support = 0.3  # Minimum support threshold
frequent_itemsets = []
items = [item for transaction in transactions for item in transaction]
itemsets = list(combinations(set(items), 1))  # Start with single itemsets

# Find frequent itemsets of size 1
for itemset in itemsets:
    support = calculate_support(itemset, transactions)
    if support >= min_support:
        frequent_itemsets.append((itemset, support))

# Debug: Print frequent itemsets of size 1
print("\nFrequent Itemsets of Size 1:")
for itemset, support in frequent_itemsets:
    print(f"Itemset: {itemset}, Support: {support:.2f}")

# Find frequent itemsets of size 2
itemsets_size_2 = list(combinations(set(items), 2))
for itemset in itemsets_size_2:
    support = calculate_support(itemset, transactions)
    if support >= min_support:
        frequent_itemsets.append((itemset, support))

# Debug: Print frequent itemsets of size 2
print("\nFrequent Itemsets of Size 2:")
for itemset, support in frequent_itemsets:
    print(f"Itemset: {itemset}, Support: {support:.2f}")

# Step 4: Generate association rules
def generate_rules(frequent_itemsets, transactions, min_confidence):
    rules = []
    for itemset, support in frequent_itemsets:
        if len(itemset) > 1:  # Only consider itemsets larger than 1 for rules
            for i in range(1, len(itemset)):
                antecedents = list(combinations(itemset, i))
                for antecedent in antecedents:
                    consequent = tuple(set(itemset) - set(antecedent))
                    # Calculate confidence
                    antecedent_support = calculate_support(antecedent, transactions)
                    if antecedent_support > 0:
                        confidence = support / antecedent_support
                        if confidence >= min_confidence:
                            rules.append((antecedent, consequent, confidence, support))
    return rules

# Step 5: Get association rules
min_confidence = 0.6  # Minimum confidence threshold
```

➢ **<u>OUTPUT:</u>**

```
Clustering Results:
   InvoiceNo      Description  Cluster
0       1001     T-SHIRT BLUE        0
1       1002      T-SHIRT RED        1
2       1003       JEANS BLUE        0
3       1004      JEANS BLACK        0
4       1005   SNEAKERS WHITE        1
5       1006   SNEAKERS BLACK        0
6       1007      HOODIE GREY        1
7       1008      HOODIE BLUE        0
8       1009        HAT WHITE        0
9       1010        HAT BLACK        0
```



```
Transactions:
['T-SHIRT BLUE']
['T-SHIRT RED']
['JEANS BLUE']
['JEANS BLACK']
['SNEAKERS WHITE']
['SNEAKERS BLACK']
['HOODIE GREY']
['HOODIE BLUE']
['HAT WHITE']
['HAT BLACK']

Frequent Itemsets of Size 1:

Frequent Itemsets of Size 2:

Association Rules:
No association rules generated.
```